Policy Transfer for Hierarchical Goal-Conditioned Reinforcement Learning

Usman Islam [1] **Zhixun Chen** [2] **Stefanos Leonardos** [1] **Matteo Leonetti** [1] **Yali Du** [1]

Abstract

Goal-Conditioned Reinforcement Learning (GCRL) tackles the challenging problem of long-horizon, sparse-reward goal-reaching, often for robotic navigation or manipulation tasks in 2D or 3D spatial domains. Recent hierarchical approaches, relying on a two-level policy along with a graph of landmarks in goal space, have demonstrated reasonable asymptotic performance. However, existing algorithms suffer from poor sample efficiency due to too many uninformative landmarks and the inability to transfer low-level behaviour between related tasks. We instead claim that transferring a pre-trained low-level policy between environments can significantly improve landmark generation, sample efficiency and even success rates. We introduce an algorithm PROMO, which uses a pre-trained low-level policy and a learned reachability model to generate landmarks that give access to large novel (previously unreachable) regions of goal space whilst remaining reachable from the previous landmark. Whereas previous works incorporate the idea of reachability/novelty only implicitly or using heuristics, we explicitly model reachability using a structured neural architecture. We demonstrate 3-4x improvements in sample efficiency over existing state-of-the-art methods on the challenging robotics tasks of AntMaze and Reacher3D, with the mild overhead of one-time policy pre-training. In addition, our method achieves superior success rates across all environments, as well as better training stability and much fewer, more informative landmarks.

1. Introduction

GCRL is a promising paradigm to solve Reinforcement Learning (RL) tasks parameterised by a desired goal. These tasks can encode far more challenging information than their standard RL counterparts, owing to the parameter-dependent, sparse reward structure. Many important spatial applications such as robotic navigation and manipulation can be expressed as goal-reaching tasks in low-dimensional goal spaces. Recently, several methods (Lee et al., 2022; Yoon et al., 2024; Park et al., 2024) have tackled this problem by building a graph of *landmarks*, i.e. important sub-goals, traversing through which gives access to much more of goal space (e.g. a crossroads). These methods employ a hierarchical structure, with a low-level policy implementing sub-goals and a high-level policy maintaining the graph and choosing sub-goals for the low-level to implement. Shortest-path algorithms are used to navigate through the graph. While these techniques have proven worthwhile, they are still inefficient in terms of environment steps needed for training and often fail to solve very long-horizon tasks.

Current methods fall short in two main ways. Firstly, they train the separate skills of *moving* (low-level sub-goal fulfillment) and *planning* (high-level sub-goal generation) together. While this may benefit expressiveness for some tasks, it means that the general ability to move through the goal space (e.g. the act of walking in a maze environment) must be re-learned for every new environment, leading to large sample complexity. Moreover, existing methods use landmark generation processes based on overly simplistic heuristics of reachability and novelty, relying instead on the sheer quantity and density of landmarks generated. Generating more landmarks per unit area/volume explored means more steps to chart the goal space and less interpretable landmark graphs.

In this work, we seek to improve on hierarchical GCRL in spatial domains (2D or 3D goal spaces) by studying whether the pre-training of a goal-conditioned low-level policy can facilitate better balance between exploration (novelty) and exploitation (reachability) when generating landmarks. We replace the usual reachability heuristics with a learned neural model which exploits geometric properties inherent in spatial domains. Moreover, the pre-trained policy can be reused across related environments, e.g. different maze topologies/sizes. We consider only the 2D/3D spatial case in this work since it covers the main GCRL robotics applications and all recent state-of-the-art baselines restrict their experiments to these domains. However, we later suggest

[1]Department of Informatics, King's College London [2]The Hong Kong University of Science and Technology. Correspondence to: Usman Islam <usman.islam@kcl.ac.uk>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. Copyright 2026 by the author(s).

a natural future direction to extend the method to abstract, high-dimensional problems. Our main contributions are:

- A new landmark generation procedure for hierarchical GCRL in spatial domains, which uses a neural reachability model and a pre-trained low-level policy to produce fewer, more spaced out landmarks than existing methods.
- An extensive set of comparative and ablative experiments demonstrating the ability of our method to greatly improve both sample efficiency and goal-reaching success over state-of-the-art methods.

To our knowledge, our method is the first hierarchical GCRL method to pre-train and transfer the low-level policy between environments.

2. Related Work

Hierarchical RL methods are categorised either as skill-based or goal-based. Skill-based methods (Sohn et al., 2018; Li et al., 2019; Tessler et al., 2017; Frans et al., 2018; Pertsch et al., 2021; Wang et al., 2024) seek to pre-train multiple "primitive" skills (policies or offline data) and then transfer them to more complex environments or scenarios. The objective is to learn a discrete set of qualitatively distinct action sequences and a high-level policy that periodically chooses which one to execute. However, these methods are not designed for the goal-reaching problem, which instead comprises *smoothly varying* goals, formulated as a Goal-Augmented Markov Decision Process (GA-MDP). To specifically tackle this case, goal-based methods exploit the strong prior of rewards being based on the distance to the goal. They employ a high-level policy to generate sub-goals and a single goal-conditioned low-level policy to execute them. Since we are concerned with solving spatial GA-MDPs rather than abstract skill discovery, we limit our discussions to the goal-based literature, also known as hierarchical GCRL.

Hierarchical GCRL methods follow the *feudal* structure of a high-level policy providing sub-goals to a low-level policy, with both policies being trained together (Vezhnevets et al., 2017). Some early examples include HAC (Levy et al., 2017), HIRO (Nachum et al., 2018) and RIS (Chane-Sane et al., 2021), where the high-level policy directly outputs a sub-goal. Other methods build a graph of static landmarks (important sub-goals) and use it to plan over long distances (Zhang et al., 2021; Kim et al., 2021; Huang et al., 2019) using a shortest path algorithm, thereby improving efficiency. The graph nodes are landmarks, and the edges between them depend on some measure of similarity or reachability. Search on the Replay Buffer (SoRB) (Eysenbach et al., 2019) takes all states from the low-level replay buffer as landmarks and constructs a graph based on them. While these methods achieved some early success, they still suffer from poor sample efficiency and rely on a uniform initial state distribution.

The current state-of-the-art methods, DHRL (Lee et al., 2022), BEAG (Yoon et al., 2024) and NGTE (Park et al., 2024) are also landmark/graph-based and improved performance on the more challenging fixed distribution case, but still suffer from poor sample efficiency and excessive landmarks. DHRL samples landmarks as achieved goals from the low-level replay buffer and focuses on decoupling the low-level and high-level time horizons, while BEAG assigns landmarks as the vertices of an adaptively refined grid. Both these methods produce many uninformative landmarks. NGTE navigates to a frontier landmark and uses it as an exploration outpost, similar to ours, but then generates landmarks randomly. The very recent method SSE (Hwang et al., 2025) uses trajectory success/failure statistics to enforce strict subgoal execution, improving the reliability of sub-goals. While this method leads to fewer landmarks, it still has accuracy issues due to over-reliance on heuristics, as described below.

All the above methods implicitly or explicitly balance reachability with novelty when generating landmarks/sub-goals. However, these quantities are represented by overly simplistic heuristics. For example, many works use the low-level value function to assess reachability (e.g. DHRL, SoRB, RIS), which is only accurate for close-by states (it is trained with one-step temporal difference errors between close-by states). This means very many close-by landmarks are generated without traversing through much of the goal space. Other methods use the empirical test of whether a given landmark *actually was* reached during exploration, without any modelling. NGTE does this on randomly sampled landmarks and BEAG makes the grid more coarse (novelty-seeking) or fine (reachability-seeking) based on whether landmarks were reached. Finally, SSE delineates the reachability frontier based on failed and partially succeeded trajectories.

By contrast, we explicitly model reachability between all (including far-apart) possible sub-goals, using a pre-trained and transferred low-level policy and a reachability model with a new trajectory-accessibility architecture. Our landmark generation procedure can balance reachability and novelty of very distant goals to generate much fewer, more spaced out and therefore more informative landmarks. In addition, our method is the first to pre-train and transfer the low-level policy to help in landmark generation and reuse the *moving* behaviour across related tasks.

3. Preliminaries

Goal-augmented MDP. We formulate the problem as a finite-horizon Goal-augmented Markov Decision Process (GA-MDP) (Sutton & Barto, 2018; Nasiriany et al., 2019), defined by a 10-tuple $M = (S, A, G, T, R, d, \phi, \gamma, \rho_S, \rho_G)$. The sets S, A and G are the state, action and goal spaces respectively. We assume that G is bounded, a common assumption made by e.g. BEAG (Yoon et al., 2024). State, action and goal spaces are usually continuous and Euclidean in GCRL, though this is not a strict requirement. We adopt a convention of denoting *achieved* goals, i.e., the agent's current position in the goal space, by g and *desired* goals, e.g. the task goal, by h. $T : S \times A \to S$ and $R : S \times A \times G \to \mathbb{R}$ is a (possibly stochastic) dynamics transition function. Additionally, the known distance function $d(g_1,\, g_2) : G \times G \to \mathbb{R}^+$ (inducing the 2-norm if spaces are Euclidean) is used to determine reward. Specifically, the reward function $R : S \times A \times G \to \mathbb{R}$ is 0 if the achieved goal is within a known threshold $\delta_r \in \mathbb{R}^+$ of the desired goal, and -1 otherwise. This sparse reward case is the most challenging configuration, on which we focus in this work, but a dense reward can be defined as the negative distance to the goal. $\gamma \in [0, 1]$ is a discount factor and ρ_S and ρ_G are the initial state distribution and the task goal distribution. At the start of each episode, an initial state and a goal are sampled from these distributions. Finally, $\phi : S \to G$ is a known mapping from states to corresponding achieved goals, usually taken as the selection of the appropriate vector elements.

Policy structure. We endow the agent with a stochastic low-level policy $\pi_{\text{LL}}(a \,|\, s, h_{\text{LL}}) : A \times S \times G \to [0, 1]$ and a deterministic high-level planner $\pi_{\text{HL}}(h_{\text{LL}} \,|\, s, h) : G \times S \times G \to \{0, 1\}$, where h_{LL} is a sub-goal passed periodically from π_{HL} to π_{LL}, giving a primitive action a for the environment.

Steps where new sub-goals are produced are called *control steps* and all steps in between them produce actions by passing the same sub-goal as at the most recent control step. The initial step of an episode is always a control step, but the rest are determined by the planner. The policies must be optimised to maximise the expected discounted *return* (discounted sum of rewards) over the task horizon.

Accessibility and reachability. Here we present two concepts which will play a central role throughout the paper.

Definition 3.1 (Accessible goal space)**.** We formally define the *accessible* goal space (or region) G_{acc} of M as the largest subset of G such that for every $g \in G_{\text{acc}}$, there exists an initial state $s_1 \sim \rho_S$, a sequence $(s_2, \ldots, s_{N+1})$ of states generated by T, some action sequence $(a_1, \ldots, a_N)$ and some desired goal sequence $(h_1, \ldots, h_N)$, satisfying $\phi(s_{N+1}) = g$. This region is intuitively the set of points in goal space that can be reached at all (e.g. corridors as opposed to wall interior points in a maze). A point within this set is called *accessible*.

Definition 3.2 (Reachable goal)**.** For any two goals $g_1,\, g_2 \in G$, we say that g_2 is *reachable* from g_1 with respect to a policy $\pi(a \,|\, s, h)$ in k steps, if the expected state sequence $(s_1, \ldots, s_{k+1})$ produced by π under T, given any initial state s_1 for which $\phi(s_1) = g_1$, satisfies $d(\phi(s_{k+1}),\, g_2) < \delta_r$. This concept assesses the ability to move from one goal to another under the action of a given policy.

Finally, we define the functions $A(g) : G \to \{0, 1\}$ and $R_\pi(g_1, g_2) : G \times G \to \{0, 1\}$ to be indicator functions for accessibility and reachability, respectively. We will model these and use them for landmark generation.

4. Method

In this section, we present our algorithm **P**rogressive **R**eachability **O**ptimisation and **MO**delling (PROMO). PROMO consists of a low-level GCRL policy π_{LL}, pre-trained in a simpler environment, and a high-level graph-based planner π_{HL}, trained in the given complex environment using π_{LL} as a fixed sub-policy. One episode in the low-level environment corresponds to one high-level control step in the main environment, though the planner may incite a new control step early if it chooses.

The training procedure has three steps: low-level policy training, trajectory model training and high-level planner training (including landmark generation). At inference time, the trajectory and accessibility models (together, the *reachability* model) are used to create a landmark graph at each control step, through which the planner finds the shortest path to the goal. Landmarks (or the end goal) are provided as sub-goals to the low-level policy. The full inference flow is presented in Figure 2 while the low-level and high-level training scenarios are shown in Figure 1.

We distinguish between two formats for providing sub-goals to π_{LL}: *relative* and *absolute* goals. For translation-invariant tasks like maze navigation, sub-goals provided by π_{HL} are relative to the achieved goal at the most recent control step, whereas they are the standard absolute goals for all other tasks. Navigation tasks are translation-invariant since the skill of moving a given displacement does not depend on the absolute starting position. The remainder of this section details the three-step training procedure and the architectural components. Note that Sections 4.5 (Reachability Model) and 4.6 (Planner) are relevant for both inference time and exploration (navigating to frontier landmarks) during training time. Full pseudocode for the planner training (Algorithm 1), the exploration procedure (Algorithm 2) and the planner inference (Algorithm 3) can be found in the Appendix.

4.1. Step 1: Low-level Policy Training

The low-level policy, π_{LL}, is trained in a simple environment to reach goals in a small box centred on the agent's initial position. We use Twin-Delayed Deep Deterministic Policy Gradient (TD3) (Fujimoto et al., 2018) with Hindsight Experience Replay (HER) (Andrychowicz et al., 2017), in line with previous work, though any GCRL algorithm could be used. Note that we start the agent at the origin if the provided sub-goals will be relative, otherwise the initial position is randomised. The environment is chosen so that:

1. Its transition function is as close as possible to that of the main environment within the latter's accessible region. For instance, both environments in Figure 1 share the same physics in the corridors, even if not when colliding with walls.
2. Its accessible region is as large as possible in the goal space.
3. The desired goal is in a small (in comparison to the main environment) region of the initial achieved goal (see Figure 1).

We describe this environment as 'simple' since a fully accessible goal space allows easy exploration without trajectories being blocked. Moreover, the desired goal is always nearby compared to goals in the main environment. Such environments are easy to construct by removing obstacles while maintaining physics. In the worst case, it can be formed as a copy of the main environment, except with the desired goal spawning in a small fixed region around the initial achieved goal. Once π_{LL} is trained, its parameters are fixed so that it can be used as a transferable black-box.

4.2. Step 2: Trajectory Model Training

We train a Long Short-Term Memory (LSTM) recurrent neural network to predict the full trajectory that π_{LL} will produce, given a goal. This will be used as the first component of our reachability model. To collect data, we run several evaluation episodes of the simple environment with the pre-trained policy π_{LL}, collecting full trajectories. We then do several epochs of supervised updates on the LSTM parameters, minimising the Mean Squared Error (MSE) between corresponding goals in the predicted and actual trajectories. Full details of the loss functions can be found in the Appendix.

4.3. Step 3: High-Level Planner Training

The planner is trained on the main environment in a round-based fashion, with new landmarks being added to the landmark set progressively. The following two steps are executed in each round:

1. **EXPLORATION**
   For each of several episodes: navigate to the most recently added reachable landmark (according to the reachability model) and then explore randomly (give π_{LL} random sub-goals), adding all achieved goals to the landmark's achieved goal buffer (store them as trajectories).

2. **TRAINING**
   For any landmark l, with buffer D_l, for whom the buffer length $|D_l|$ has reached a fixed threshold:

   - Train the accessibility model using all data from D_l.
   - Generate a new landmark l_{next} from l, initialising $D_{l_{\text{next}}}$.

If the initial achieved goal of any episode cannot reach any current landmarks (e.g. at the start, when there are no landmarks), we add the goal itself as a new landmark. Whenever a new landmark is added, it is always marked *unfinished* and we initialise a new buffer for it. The algorithm terminates when all landmarks are marked *finished*. A landmark is marked finished when exploring from it no longer gives novel achieved goals. Next, we specify how novelty and reachability are balanced in landmark generation.

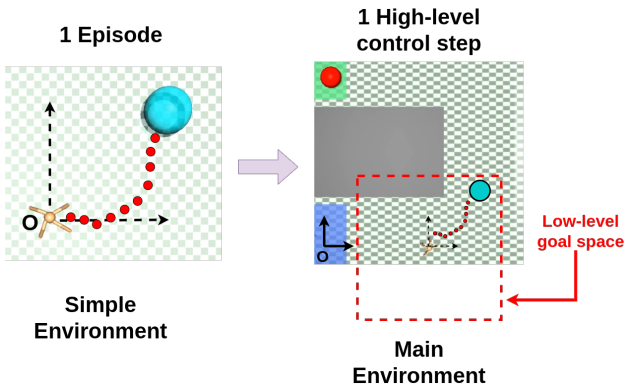


Figure 1. The agent is pre-trained to reach nearby goals in a small environment, which is free of obstacles, within a dedicated goal space (the low-level goal space). The policy is then transferred to the main environment, where the high-level planner gives the low-level policy nearby sub-goals toward the overall task goal. For relative goals as in AntMaze, note how the low-level goal space can be superimposed onto the main environment's goal space to determine the valid area around the current achieved goal within which the trained low-level policy will be able to operate effectively. For absolute goals, the low-level goal space is equal to the main environment's goal space.

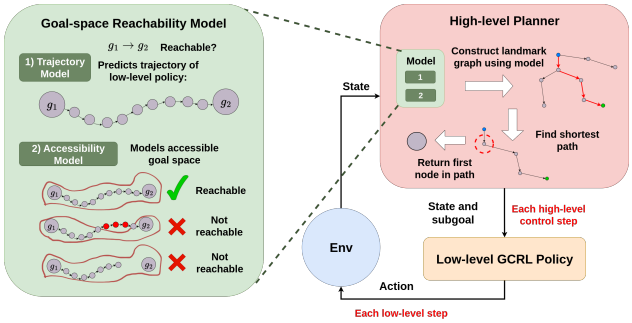


Figure 2. PROMO architecture and inference flow. The reachability model (comprising the trajectory and accessibility models) facilitates construction of the landmark graph at each control step. Some examples of reachable/unreachable goals are shown. Note how the trajectory must reach far enough (so that the trajectory model predicts success in the low-level goal space) and be fully contained within the accessible region (according to the accessibility model).

4.4. Landmark Generation

We now detail the landmark generation procedure during planner training. To generate a new landmark from a given landmark and a list D_l of achieved goals collected during exploration, we discretise the goal space into a high-dimensional grid of bins (formally hyper-boxes in G) and assign a score for each bin. The new landmark is the centroid of the bin with the highest score. We additionally produce a few backup landmarks by treating the score function as a probability distribution and sampling from it.

The score function is a product of three components, enforcing *novelty*, *reachability* and *boundary separation* respectively. We express the score $\mathcal{S}_\mathcal{B}$ for bin $\mathcal{B}$ as:

$$\mathcal{S}_\mathcal{B} := \mathcal{N}_\mathcal{B}\, \mathcal{R}_{\mathcal{B},\, l}\, \mathcal{G}(\mathcal{A}_\mathcal{B})\,, \tag{1}$$

where the respective terms quantify how much access a landmark gives to novel areas, how likely it is to be reachable from the previous landmark and whether it has a sufficient separation from the boundary of the accessible region. We explain these quantities in detail next.

Novelty Access Score $\mathcal{N}_\mathcal{B}$ is a measure of how well a given achieved goal gives access to novel parts of the goal space, if navigating through it using π_{LL}. This is determined by measuring the proportion of novel achieved goals in trajectories outgoing from the given achieved goal, with novelty being defined as non-reachability from all existing landmarks according to the reachability model.

Formally, let g_i be a goal from D_l. We implement the buffer D_l so that we have access to the full trajectory of g_i. Define $(g_i^1,\, ...,\, g_i^{m_i})$ as the remaining segment of g_i's trajectory *after* g_i, for some $m_i \in \mathbb{N}$. Then, the novelty access score is the (bin-wise) average proportion of future goals in the trajectory which are novel:

$$\begin{aligned} \mathcal{N}_\mathcal{B} &:= \frac{1}{n_\mathcal{B}} \sum_{\substack{i \\ \text{s.t. } g_i \in \mathcal{B}}} \mathcal{N}(i), \\ \text{where} \quad \mathcal{N}(i) &:= \frac{\sum_{j=1}^{m_i} \mathbb{I}(N_{\pi_{\text{LL}},\, l}(g_i^j) = 1)}{m_i}\,. \end{aligned} \tag{2}$$

Here, $n_\mathcal{B}$ is the number of goals from D_l lying in bin $\mathcal{B}$ and $\mathcal{N}(i)$ is a novelty access score for the i-th goal in D_l, defined in terms of the indicator function $\mathbb{I}(\cdot)$ and the following function quantifying the novelty of a goal:

$$N_{\pi_{\text{LL}},L}(g) := \prod_{n=1}^{|L|} (1 - R_{\pi_{\text{LL}}}(l_n,\, g))\,. \tag{3}$$

$N_{\pi_{\text{LL}},L}(g)$ depends on the current set L of landmarks and assumes access to the reachability function $R_{\pi_{\text{LL}}}(g_1,\, g_2) : G \times G \rightarrow \{0, 1\}$ which we will model in the next section. $N_{\pi_{\text{LL}},L}(g)$ has value 1 if the input goal is *not* reachable from any current landmark and 0 otherwise.

Reachability Score Next, we present the bin-wise *reachability score*, representing the probability that the centroid of the bin is reachable from the exploration landmark l according to our reachability function $R_{\pi_{\text{LL}}}(g_1,\, g_2)$:

$$\mathcal{R}_{\mathcal{B},\, l} := R_{\pi_{\text{LL}}}(l,\, c_\mathcal{B})\,. \tag{4}$$

This is needed to make sure the new landmark can actually be reached during planning.

Boundary Separation Score Finally, we introduce a component score which makes sure that landmarks are not generated very close to the edge of the accessible region of the goal space, since this can lead to reachability problems during planning. The score is a Gaussian-smoothed measure of the average accessibility of achieved goals lying within the bin. Consider the bin-wise accessibility score:

$$\mathcal{A}_\mathcal{B} := \frac{1}{n_\mathcal{B}} \sum_{\substack{i \\ \text{s.t. } g_i \in \mathcal{B}}} A(g_i) \quad , \tag{5}$$

where $A(g)$ (not to be confused with the action space A) is our accessibility function, also modelled in the next section. This measures, on average, whether the goals lying in the bin are accessible. We put this through a Gaussian filter (Haddad et al., 1991) to smooth the boundaries between high and low scores, thereby lowering the score just inside the

boundary. Let $\mathcal{G}$ represent the Gaussian filter convolution of $\mathcal{A}_\mathcal{B}$ (we use unit variance). This convolution is applied to the accessibility score tensor over bins $\mathcal{B}$. Then the boundary separation score component $\mathcal{G}(\mathcal{A}_\mathcal{B})$ is a factor in the final score, defined in Equation 1.

4.5. Reachability Model

Our reachability model facilitates both graph construction during inference and exploration, as well as reachability/novelty optimisation during landmark generation at training time. It consists of two components: a trajectory model predicting the full trajectory taken by π_{LL} given a goal in the simple environment, and an accessibility model which predicts whether a goal is in the accessible region of the main environment. Given two goals $g_1, g_2 \in G$, we then say g_2 is reachable from g_1 with respect to π_{LL} if and only if the following two conditions are satisfied:

1. π_{LL}'s predicted trajectory successfully reaches g_2 in the simple environment

2. The full predicted trajectory is contained in the accessible region of the main environment.

Suitability This is a good approximation since it captures the intuition that an agent needs to 1) be able to travel far enough even without obstructions and 2) have a clear (accessible) path to the goal. While these conditions are usually not enough to *guarantee* reachability in practice, we find that, if used with a correction mechanism (which we describe later), it is enough to achieve excellent results. See Figure 2 for a visualisation of the two conditions. The trajectory model training procedure has been discussed earlier, and all that is left is to train the accessibility model.

Accessibility Model In each round, the accessibility model receives a new batch of goals achieved during exploration and must learn to bound the global distribution of all rounds' data. Any batch-trained, unsupervised density estimator or one-class classifier can be used. The output is a binary value representing inclusion in the learned accessible region. Some example methods are (deep) one-class classifiers (Seliya et al., 2021), deep autoencoders (Zong et al., 2018) and normalising flows (Kobyzev et al., 2020). In our experiments, we use an ensemble of one-class Support Vector Machine (SVM) classifiers (more details in Appendix B), for their ease of implementation, but any method could be substituted. Before high-level training, the model is initially trained on several random exploration episodes. Each subsequent training round naturally fine-tunes the model on data near the chosen exploration landmark.

4.6. Planner

At each control step during inference (or exploration), our planner π_{HL} selects a sub-goal for π_{LL} by finding the shortest path through a graph of the current landmarks (plus the current achieved goal and desired goal) using Dijkstra's algorithm and then choosing the first node in the path. Nodes are connected based on reachability. A (unit weight) edge exists between nodes n_i and n_j of the graph if and only if at least one of the following three conditions are met:

$$\begin{aligned} &1) \quad i = j \\ &2) \quad n_i \text{ and } n_j \text{ are landmarks and } n_j \text{ was generated from } n_i \\ &3) \quad R_{\pi_{\text{LL}}}(n_i, n_j) = 1 \ . \end{aligned} \tag{6}$$

Due to errors in the reachability model, it is possible (though rare) for π_{HL} not to reach a given node. To mitigate this, we introduce a simple correction mechanism for planning. If a goal is not reached for two consecutive high-level control steps, π_{HL} instead provides a random nearby goal to π_{LL} in the next control step.

4.7. Algorithm Termination

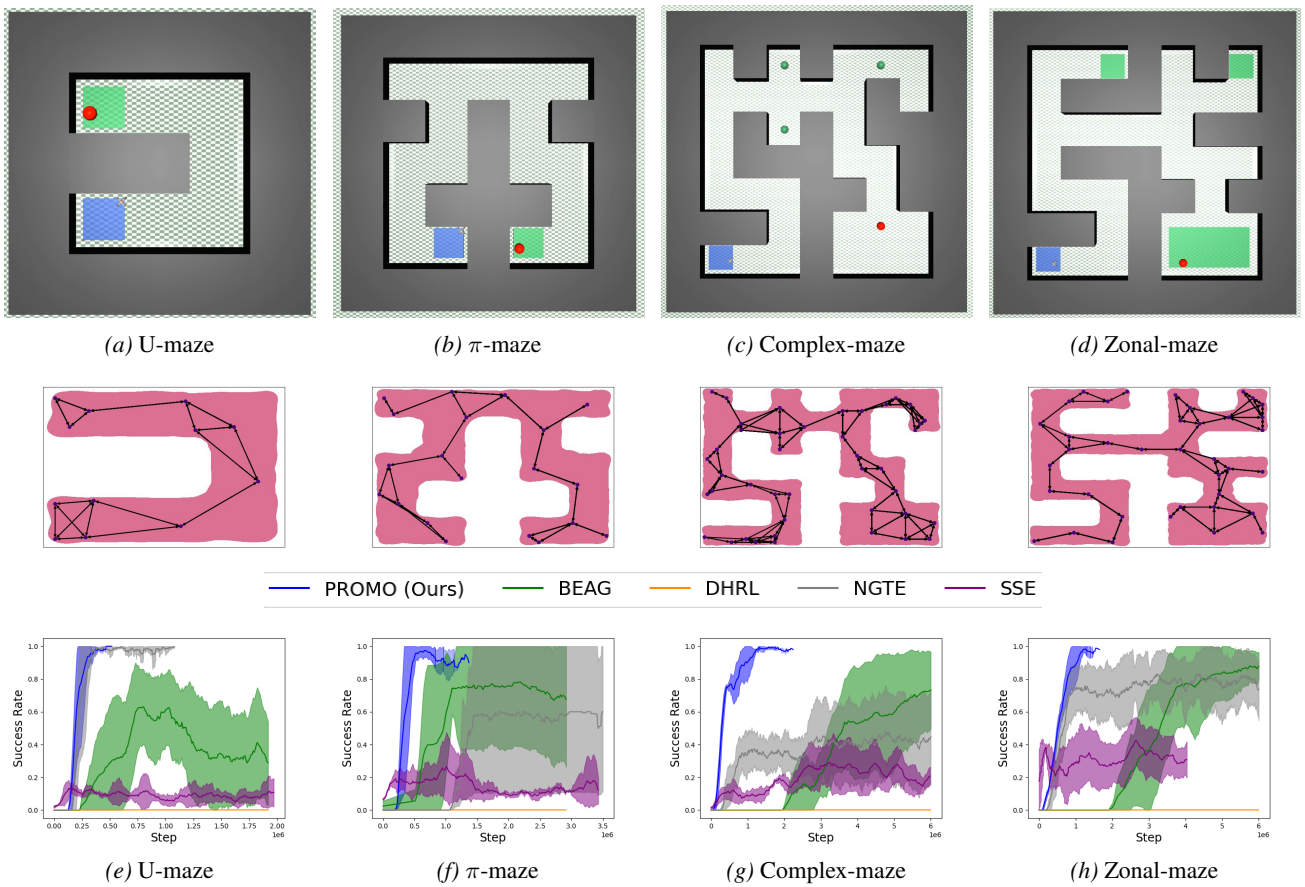
The planner training algorithm terminates when all landmarks have been marked *finished*. Each round, we finish the landmark from which exploration was done if the number of novel goals reachable by the would-be next landmark, as a proportion of the round's collected achieved goals, is smaller than a threshold ϵ. We use the value $\epsilon = 0.001$ throughout our experiments. Mathematically, this finishing condition can be expressed as follows, given an exploration landmark l, a candidate next landmark l_{next} and a list D_l of goals achieved during exploration:

$$\frac{|\{g \in D_l : N_{\pi_{\text{LL}},\, L}(g) = R_{\pi_{\text{LL}}}(l_{\text{next}}, g) = 1\}|}{|D_l|} < \epsilon \ , \tag{7}$$

where $N_{\text{LL},\, L}$ and $R_{\pi_{\text{LL}}}$ are the novelty and reachability functions presented earlier. If this criterion is satisfied, l_{next} is discarded and l is marked finished. Otherwise, l_{next} is added to the set of landmarks. Note that, l_{next} is only added if a trial episode shows that it is actually reachable from l, otherwise the backups are tested for this. If no backups succeed, no landmark is added but l is not finished.

5. Experiments

We tested our method on challenging new and existing instantiations of the two standard MuJoCo benchmarks for GCRL: AntMaze, a 2D quadruped maze navigation task and



(a) U-maze *(b)* π-maze *(c)* Complex-maze *(d)* Zonal-maze

(e) U-maze *(f)* π-maze *(g)* Complex-maze *(h)* Zonal-maze

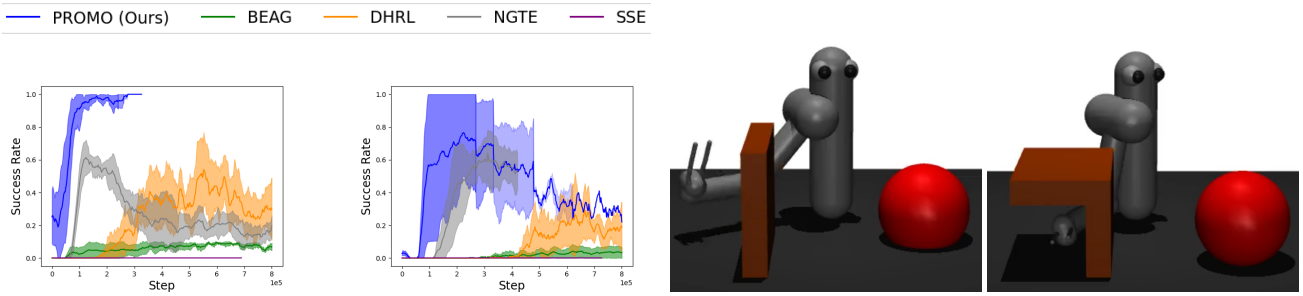
Figure 3. Visualisations (first row), generated landmark graph with predicted accessible region (second row) and results (third row) for the AntMaze environments, with initial distributions in blue and goal distributions in green. The low-level policy and model were pre-trained for 2.7M total steps (pre-training curves in Appendix D), and then reused for each of the above environments. Our algorithm and all baselines were trained over 5 seeds, except SSE (3 seeds).

Reacher, a 3D robot arm spatial goal-reaching task. These are visualised in Figure 3 and full environment details are given in Appendix C. Throughout our experiments, we used sparse reward and the initial state was fixed (with some small local variation). All experiments were evaluated by the average success rate of goal-reaching over 10 evaluation episodes periodically through training, where an episode is considered a success if the goal is reached at any point in its duration. Next, we present comparative results, while full details of ablative studies, computational details and wall clock time tests are given in Appendices E and G. In particular, our wall clock time experiment demonstrated that the pre-training does not add any extra time per environment step, allowing us to gauge the time saved by pre-training simply by looking at the environment steps in comparison with the baselines.

5.1. Comparative Results

In this section, we compare our method to the four state-of-the-art graph-based, hierarchical GCRL baselines mentioned earlier: DHRL (Lee et al., 2022), BEAG (Yoon et al., 2024), NGTE (Park et al., 2024) and SSE (Hwang et al., 2025). We first compare the success rates and sample efficiency and then show that merely transferring a low-level policy does not improve the baselines' results, therefore demonstrating how both levels of our algorithm work *together* to produce better results. Finally, note that our algorithm produces significantly fewer and therefore more informative landmarks, thus exploring the space faster. Figure 9 shows some examples of the landmark graphs produced by DHRL, BEAG and NGTE (taken from their papers) alongside those produced by our algorithm.

Success Curves As shown by Figures 3 and 4, our algorithm yields significant improvements not only in sample efficiency, but also in success rates and training stability throughout all experiments. In all environments tested, our algorithm 1) reached its peak performance faster than any other baseline (often by a factor of 3-4x), 2) achieved a near-100% success rate in all but one environment and 3) achieved a substantially smaller standard deviation between



(a) Reacher-wall *(b)* Reacher-L-wall *(c)* Reacher-wall *(d)* Reacher-L-wall

Figure 4. Results for Reacher environments. The low-level policy and model were pre-trained for a total of 1M steps (Appendix D). Our algorithm and all baselines were trained over 5 seeds, except SSE (3 seeds).

different seeds. It is worth bearing in mind that, unlike the other methods, ours requires a pre-training stage which takes up a noticeable number of steps. However, this need only be performed once and then all environments can benefit from improved sample efficiency.

Pre-trained Transfer in the Baselines To dispel the idea that our method simply achieves good results by virtue of a simple change in the structure of training, we ran the two best performing baselines (BEAG and NGTE) with *our* pre-trained low-level policy, using relative sub-goals and frozen parameters, on π-maze and Zonal-maze. If indeed all our improvements were simply due to pre-training, we would expect the baselines to match our results here. However, both baselines were not able to effectively utilise the policy for either maze, as shown in Figure 10. For example, BEAG only managed to explore half of Zonal-maze. Figure 10d shows a high-level plan generated based on this incomplete exploration. This failure to adapt is expected, since current hierarchical GCRL methods use end-to-end joint training. This cannot avoid specialising the low-level policy to the particular environment/maze.

6. Limitations and Future Work

We note that, just like all the state-of-the-art baselines (SSE, NGTE, BEAG, DHRL), our algorithm has only been tested on low-dimensional spatial robotics tasks. While our method effectively exploits geometry to solve these, more work is needed to extend it to cases where dimensions are high or dynamics do not follow smooth physical laws (e.g. when teleportation is allowed). In this vein, we intend in future work to replace the trajectory-accessibility architecture with a single neural network that outputs the estimate of reachability for a pre-trained low-level policy being applied in a complex environment. This is conceptually simpler and more general but its naive form would require the collection of data from sufficiently many starting goals to sufficiently many ending goals. Solving this quadratic exploration complexity problem would allow our method to apply to any GCRL method of any dimensionality, as well as relax the requirement for the simple and complex environment dynamics to strongly match. Note that, for higher dimensions, the landmark generation would need to be adjusted so that landmarks are selected from the set of visited achieved goals, rather than the full grid, whose size increases exponentially with dimension.

7. Conclusion

In summary, we have presented a new hierarchical GCRL algorithm which uses pre-training and reachability modelling to improve sample efficiency and performance on challenging maze and robot arm environments, compared to several state-of-the-art baselines. Our goal was to examine whether transfer-augmented reachability modelling could provide benefits to GCRL, especially in terms of sample efficiency. Through our experiments, we have answered this question overwhelmingly in the affirmative. Moreover, we even achieved significantly better overall performance and stability than the previous state-of-the-art methods. We hope that this work will stimulate further interest in the broader applications of transfer learning in RL.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Chane-Sane, E., Schmid, C., and Laptev, I. Goal-conditioned reinforcement learning with imagined sub-

goals. In *International Conference on Machine Learning*, pp. 1430–1440. PMLR, 2021.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. Meta learning shared hierarchies. In *International Conference on Learning Representations*, 2018.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Haddad, R. A., Akansu, A. N., et al. A class of fast gaussian binomial filters for speech and image processing. *IEEE Transactions on Signal Processing*, 39(3):723–727, 1991.

Huang, Z., Liu, F., and Su, H. Mapping state space using landmarks for universal goal reaching. *Advances in Neural Information Processing Systems*, 32, 2019.

Hwang, J., Lee, S., Kim, J., and Han, S. Strict subgoal execution: Reliable long-horizon planning in hierarchical reinforcement learning. *arXiv preprint arXiv:2506.21039*, 2025.

Kim, J., Seo, Y., and Shin, J. Landmark-guided subgoal generation in hierarchical reinforcement learning. *Advances in neural information processing systems*, 34: 28336–28349, 2021.

Kobyzev, I., Prince, S. J., and Brubaker, M. A. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979, 2020.

Lee, S., Kim, J., Jang, I., and Kim, H. J. Dhrl: a graph-based approach for long-horizon and sparse hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 35:13668–13678, 2022.

Levy, A., Jr., R. P., and Saenko, K. Hierarchical actor-critic. *CoRR*, abs/1712.00948, 2017. URL http://arxiv.org/abs/1712.00948.

Li, S., Wang, R., Tang, M., and Zhang, C. Hierarchical reinforcement learning with advantage-based auxiliary rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Nasiriany, S., Pong, V., Lin, S., and Levine, S. Planning with goal-conditioned policies. *Advances in Neural Information Processing Systems*, 32, 2019.

Park, J., Oh, S., and Kim, Y. Novelty-aware graph traversal and expansion for hierarchical reinforcement learning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 1846–1855, 2024.

Pertsch, K., Lee, Y., and Lim, J. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pp. 188–204. PMLR, 2021.

Seliya, N., Abdollah Zadeh, A., and Khoshgoftaar, T. M. A literature review on one-class classification and its potential applications in big data. *Journal of Big Data*, 8 (1):122, 2021.

Sohn, S., Oh, J., and Lee, H. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *Advances in neural information processing systems*, 31, 2018.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D., and Mannor, S. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI conference on artificial intelligence*, 2017.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In *International conference on machine learning*, pp. 3540–3549. PMLR, 2017.

Wang, Z., Hu, J., Chuck, C., Chen, S., Martín-Martín, R., Zhang, A., Niekum, S., and Stone, P. Skild: Unsupervised skill discovery guided by factor interactions. *Advances in Neural Information Processing Systems*, 37:77748–77776, 2024.

Yoon, Y., Lee, G., Ahn, S., and Ok, J. Breadth-first exploration on adaptive grid for reinforcement learning. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 57331–57349. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/yoon24d.html.

Zhang, L., Yang, G., and Stadie, B. C. World model as a graph: Learning latent landmarks for planning. In *International conference on machine learning*, pp. 12611–12620. PMLR, 2021.

Zong, B., Song, Q., Min, M. R., Cheng, W., Lumezanu, C., Cho, D., and Chen, H. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJJLHbb0-.

A. Trajectory Model Training

Here, we present the loss function for trajectory model training. For relative goals, the trajectory model is a function $F_{\pi_{\text{LL}}} : G' \to \bigtimes_{i=1}^{k} G'$ of one (desired) goal, since the starting goal is always the origin. For absolute goals, the function is $F_{\pi_{\text{LL}}} : G' \times G' \to \bigtimes_{i=1}^{k} G'$ as an additional starting goal input must be provided. We do several episodes of inference in the simple environment and train a Long Short-Term Memory (LSTM) recurrent neural network with k sequential cells to model $F_{\pi_{\text{LL}}}$ by minimising the discrepancy between predicted and actual trajectories. Initial state is randomised, provided the initial achieved goal is the origin if using relative goals. Every cell's input is identical. Given a list $(\tau_1, ..., \tau_N)$ of N length-k achieved goal trajectories, a corresponding list $(g_1, ..., g_N)$ of initial achieved goals and a corresponding list $(h_1, ..., h_N)$ of desired goals, the mean squared error (MSE) loss is:

$$\mathcal{L}_{\text{traj}}^{\text{rel}} := \frac{1}{kN} \sum_{i=1}^{N} \sum_{j=1}^{k} \left\| \tau_i^j - F_{\pi_{\text{LL}}}(h_i)_j \right\| , \tag{8}$$

for relative goals, and

$$\mathcal{L}_{\text{traj}}^{\text{abs}} := \frac{1}{kN} \sum_{i=1}^{N} \sum_{j=1}^{k} \left\| g_i^j - F_{\pi_{\text{LL}}}(g_i, h_i)_j \right\| , \tag{9}$$

for absolute goals. Here, we have used the notation τ_i^j to mean the jth goal in the ith trajectory of $(\tau_1, ..., \tau_N)$. Given the success threshold δ_r and the accessibility function $A(g)$, the reachability function is then expressed as:

$$R_{\pi_{\text{LL}}}(g_1, g_2) = \begin{cases} & \text{if} \quad \| F_{\pi_{\text{LL}}}(g_2 - g_1)_k - (g_2 - g_1) \| < \delta_{\text{r}} \\ 1, & \text{and} \quad A(F_{\pi_{\text{LL}}}(g_2 - g_1)_i + g_1) = 1 \\ & \forall\, i \in \{1, ..., k\} , \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

for relative goals, and

$$R_{\pi_{\text{LL}}}(g_1, g_2) = \begin{cases} & \text{if} \quad \| F_{\pi_{\text{LL}}}(g_1, g_2)_k - g_2 \| < \delta_{\text{r}} \\ 1, & \text{and} \quad A(F_{\pi_{\text{LL}}}(g_1, g_2)_i) = 1 \\ & \forall\, i \in \{1, ..., k\} , \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

for absolute goals.

B. Accessibility Model Training

For the accessibility model, we trained a Support Vector Machine (SVM) classifier which uses a data batch of collected achieved goals to bound a region of the goal space as the accessible region, classifying inclusion in it. Since we only have access to positive data points (achieved goals), we use a *one-class* version of SVM. Additionally, since the data is collected progressively, we train a new SVM each round, on only that round's exploration data, and consider the overall region to be the union of the regions modelled by the individual components. Therefore, the model is actually an *ensemble* of one-class SVM classifiers. Since each round's data is directly tied to a new base SVM, the initial exploration (before high-level training) is neither possible nor needed with this approach (as it would be for a single global model). We chose this method since it can bound relatively complex data distributions while being easy to implement with `scipy`. However, for high-dimensional goal spaces, a neural method such as normalising flows or neural autoencoders would be a better choice.

Finally, to speed up inference, our implementation stores a bounding box for the data used to train each base SVM. When querying inclusion of a point in the union, the bounding boxes are first checked (a much faster operation) and then only the

base SVMs corresponding to the containing bounding boxes are checked. This trick greatly improves inference speed for large goal spaces like AntMaze, but would not be needed for a single, global accessibility model.

C. Environments

C.1. AntMaze

AntMaze consists of a robot ant quadruped which must navigate a maze to the end goal. All mazes were trained and evaluated with a success threshold of 1 (in contrast to previous works which evaluate on a more lenient value of 5). The goal is sampled uniformly from either a small zone, multiple small zones or multiple points. The goal distribution is kept the same for training and evaluation. Our simple environment was set to a 20 x 20 wall-free box centred on the origin, with training success threshold 1. The low-level policy was trained once per seed for 2.5M steps, with the trajectory model trained for a further 200K steps. It was then transferred to the following four mazes:

- U-maze: 24 x 24, 600 steps
- π-maze: 40 x 40, 1200 steps
- Complex-maze: 56 x 56, 2000 steps
- Zonal-maze: 56 x 56, 2500 steps.

C.2. Reacher

Whereas AntMaze's goal space is 2-dimensional, we consider two 7-DoF robot arm environments, where the end effector must reach a given goal in 3-dimensional space. The first, Reacher-wall, contains a simple vertical wall to cross over whereas the second, Reacher-L-wall, contains two walls that form an L, where the end effector starts in an area trapped by the walls. In both environments, the initial position is randomised in a small box and the goal distribution is constrained (both during training and evaluation) to a small box on the other side of the obstacle. We use a training success threshold of 0.1 and an evaluation success threshold of 0.2. To pre-train our low-level policy, the simple environment has the same dimensions as the main environments but does not contain any walls. Here, the initial position is randomised and the goal is sampled from a small box ($0.5 \times 0.5 \times 0.5$) centred on the initial position. The arm's reach (true accessible region without obstacles) is roughly 0.8 units either side and about 0.6 units forward (though it is curved) We trained both the low-level policy and trajectory model for 500K steps each.

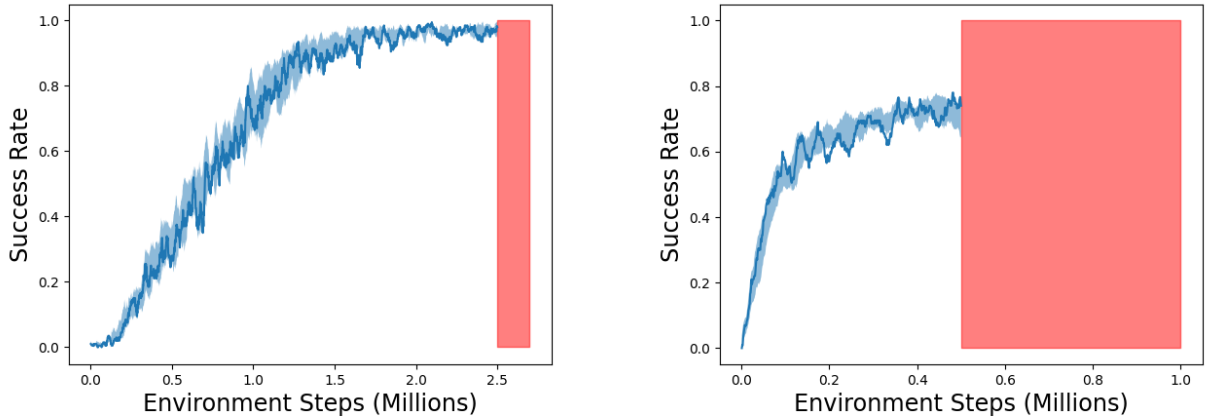
D. Low-level Training Curves

We trained the low-level policy for 2.5M steps on our simple AntMaze environment assuming relative goals. We then trained the trajectory model for 200K steps. For Reacher, both the policy and model were trained for 500K steps each. Since the trajectory model training consists of offline data collection, followed by several epochs of updates, we cannot show trajectory model success curve against environment steps. Instead we simply shade in the appropriate number of steps after the policy training curve. The curves for both AntMaze and Reacher are shown in Figure 5, averaged over 5 seeds.

E. Ablations

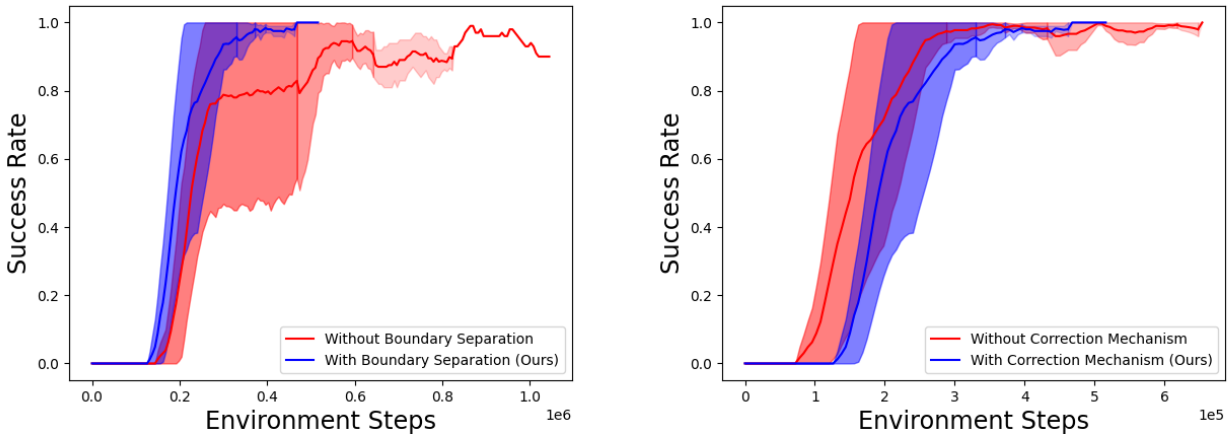
To show robustness and component contribution, we provide the following ablations in AntMaze:

1. Without boundary separation term in landmark generation (U-maze, testing the term's contribution). **Result:** performance, efficiency and stability noticeably degraded. Figure 6a
2. Without correction mechanism (U-maze and π-maze, testing the mechanism's contribution). **Result:** U-maze shows no significant effect but π-maze shows some degradation in stability as well as more steps to terminate. Figures 6b and 7a
3. With action noise added to π_{LL} in high-level training/evaluation (U-maze, testing robustness to both stochasticity and mismatched high/low-level transition dynamics). **Result:** robust performance and efficiency well maintained, though large action noise values seem to slightly increase the time to terminate. Figure 7b



(a) AntMaze *(b)* Reacher

Figure 5. Low-level policy training curves, averaged over 5 seeds and smoothed. Red shaded area shows the additional steps used for trajectory model training.



(a) Boundary Separation *(b)* Correction Mechanism (U-maze)

Figure 6. Ablative Results

4. Varying landmark termination threshold ϵ (U-maze, examining hyperparameter-sensitivity). **Result:** smaller values give good results whereas larger values cause early termination, as expected. Figure 8a

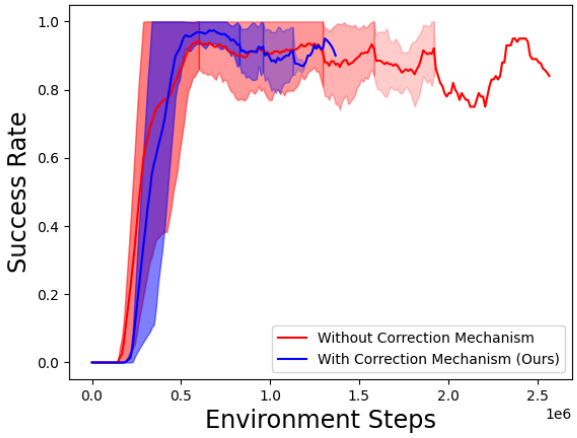
The results are given in Figures 6, 7 and 8.

F. Landmarks

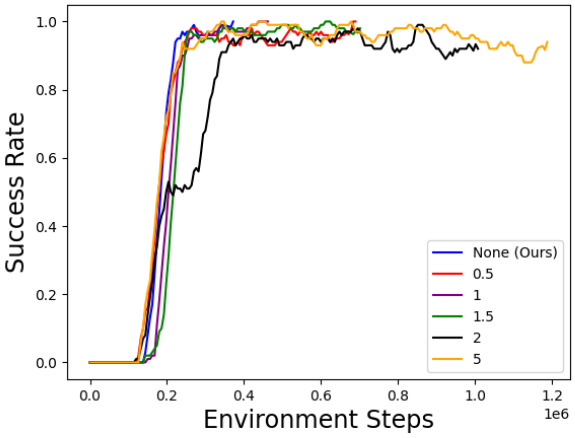
Figure 9 shows visual comparisons of the landmarks generated by BEAG and DHRL (figures presented in their papers) with our landmarks.

G. Computation and Wall Clock Time

We ran our main experiments on a commodity CPU node with 8 cores and 24GB of memory. Additionally, in order to demonstrate that the computational overhead of pre-training is not disproportionate to high-level training, we performed timed runs for all three algorithm stages with U-maze on a single machine, a Dell Intel I7 laptop with 16.5GB of memory.

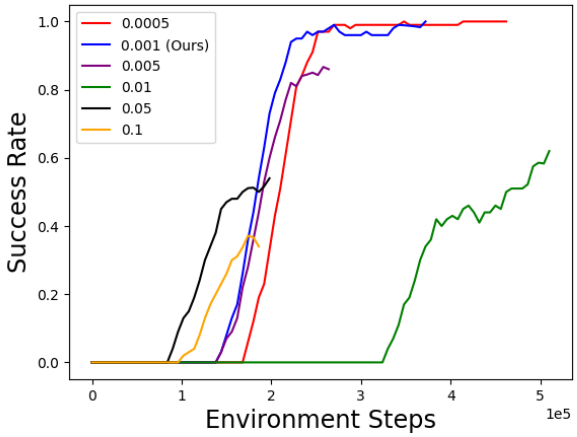


(a) Correction Mechanism (π-maze)



(b) Gaussian action noise with different scales

Figure 7. Ablative Results

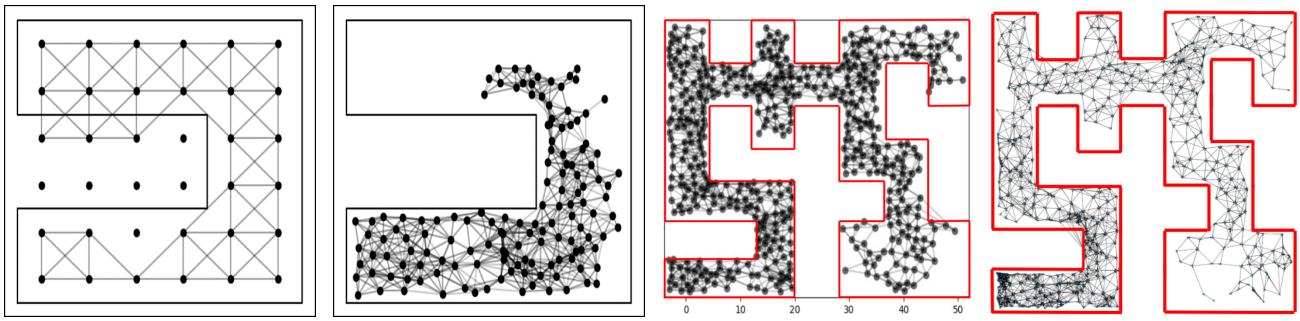


(a) Varying the termination parameter ϵ

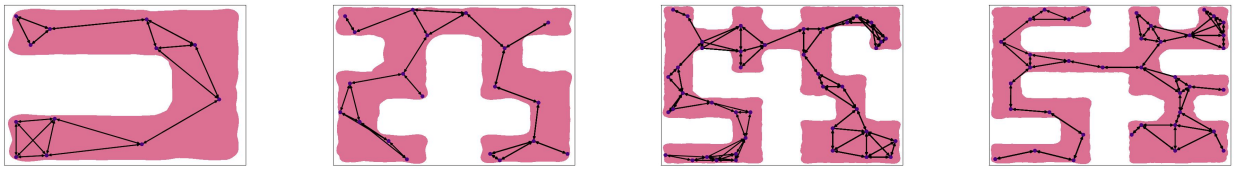
Figure 8. Ablative Results / Statistical Investigation

This was chosen since our cluster produces very unpredictable speeds. The times were then divided by the total number of environment steps for the corresponding stage and the time-per-step values were compared. In particular, policy pre-training took $37,271,471$ ms (≈ 10.4 hrs) for $2,500,000$ steps, trajectory model training took $1,768,500$ ms (≈ 29.5 min) for $200,000$ steps and the planner took $12,145,708$ ms (≈ 3.4 hrs) for $739,349$ steps. The time-per-step rates are therefore 15, 9 and 16 milliseconds per step respectively. This shows that there is no increased computational burden for pre-training, up to the number of environment steps used.

Since the pre-training time-per-step rates are similar to or lower than planner training, we can use the number of environment steps as a proxy to compare overall computational burdens (at least to rule out added overhead based on our pre-training contribution). As shown in our comparative results, our method significantly improves sample efficiency if given a one-time pre-training budget. For the price of 2.7M pre-training steps, our algorithm saves around 3M steps in both Complex-maze and Zonal-maze and attains significantly better success rates in Reacher, thus justifying the trade-off.

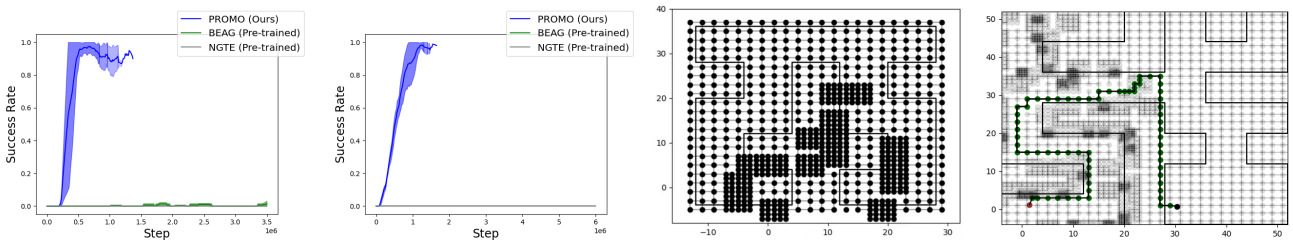


(a) BEAG U *(b)* DHRL U *(c)* DHRL Complex *(d)* NGTE Complex



(e) PROMO U *(f)* PROMO π *(g)* PROMO Complex *(h)* PROMO Zonal

Figure 9. Various AntMaze environment landmark graphs visualised for DHRL, BEAG, NGTE and PROMO (ours).



(a) π-maze *(b)* Zonal-maze *(c)* BEAG π Landmarks *(d)* BEAG Zonal Plan

Figure 10. Left two: baselines cannot adapt our pretrained (fixed) low-level policy to AntMaze tasks. Right two: visualisations of graphs generated by the BEAG baseline with our pre-trained low-level policy. Note the generated plan in Zonal-maze goes through the walls since only half of the maze has been explored.

H. Transfer in the Baselines (Investigation)

To prove that our comparative results cannot be matched simply by incorporating transfer into existing methods, we applied our pre-trained policy to BEAG and NGTE. As expected, this returned poor results, shown in Figure 10.

I. Pseudocode and Hyperparameters

Algorithm 1 PROMO Planner Training

Require: Steps per round n_{steps}, pre-trained low-level policy π_{LL}, maximum control step interval k, initial state distribution ρ_S, low-level goal space G_{LL}, environment `env`, state-goal mapping ϕ, train threshold n_{tr}

Train accessibility model on initial exploration data (explore directly from initial states)
Landmark set $L \leftarrow \{\}$
Buffers $\{D_i\}_i \leftarrow$ `List()`
for $r = 1, 2, \ldots$ **do**
  $L, \{D_i\}_i =$ `Explore(`L`,` $\{D_i\}_i$`)`
  for $l \in L$ **do**
    if $|D_l| \geq n_{\text{tr}}$ **then**
      Train Accessibility Model using D_l
      Calculate $\mathcal{S}_{\mathcal{B}}$ over grid bins $\mathcal{B}$
      $l_{\text{max}} \leftarrow \text{argmax}_{\mathcal{B}}\, \mathcal{S}_{\mathcal{B}}$
      Check finishing condition for l using l_{max}
      if Finishing condition satisfied **then**
        Discard l_{max}
        Mark l **finished**
      end if
      Sample q backups $b_1, \ldots, b_q$ as bin centroids (with bin $\mathcal{B}$ probability $\frac{\mathcal{S}_{\mathcal{B}}}{\sum_{\mathcal{B}} \mathcal{S}_{\mathcal{B}}}$)
      for Candidate l_{next} in $(l_{\text{max}}, b_1, \ldots, b_q)$ **do**
        Trial episode to verify reachability of l_{next} from l
        if verified **then**
          Add l_{next} to L
          Initialise $D_{l_{\text{next}}}$
          break
        else
          Discard l_{next}
        end if
      end for
    end if
  end for
  if All landmarks **finished then**
    break
  end if
end for

Return: L

Table 1. **PROMO Hyperparameters (AntMaze)**

	Final Value	**Range Tried**
n_{steps} (expl. steps per episode)	10000	2000 - 16000
n_{bins} (landmark gen grid)	50	50
ϵ (finishing condition)	0.001	0.001, 0.005
SVM RBF kernel ν	0.01	0.01 - 0.1
SVM RBF kernel γ	0.1	0.01 - 0.1
LSTM hidden dim	64	64, 128, 256
LSTM batch size	500	200, 500, 1000
LSTM num train epochs	3000	1000, 3000, 10000
LSTM train steps	200K	200K
π_{LL} Gaussian action noise σ	2.	0.1 - 2
π_{LL} learning rate	0.001	0.001
π_{LL} batch size	256	256
π_{LL} learning rate	0.001	0.001
π_{LL} train steps	2.5M	2.5M, 3M

Algorithm 2 Explore (Subroutine)

Input: L, $\{D_i\}_i$
Parameters: Landmark set L, landmark buffers $\{D_i\}_i$
step $\leftarrow 0$
Initial state $s \sim \rho_S$
while step $< n_{\text{steps}}$ **do**
  $l \leftarrow$ most recent reachable landmark
  if l is undefined **then**
    $l \leftarrow \phi(s)$
    Add l to L
  end if
  Append $\phi(s)$ to D_l
  while episode not terminated **do**
    if control step **then**
      if reached l this episode **then**
        Sub-goal $h \sim$ `Uniform`(G_{LL})
      else
        Sub-goal $h \leftarrow$ `Plan`$(\phi(s),\ l_e,\ L)$
      end if
    end if
    Low-level action $a \leftarrow \pi_{\text{LL}}(s,\ h)$
    Next state $s \leftarrow$ `env.step`(a)
    Add $\phi(s)$ to D_l
  end while
end while
Return: L, $\{D_i\}_i$

Algorithm 3 Plan (Subroutine)

Require: Reachability function $R_{\pi_{\text{LL}}}$
Input: g, h, L
Parameters: Achieved goal g, desired goal h, landmark set L
Construct landmark graph
Add g, h to graph using $R_{\pi_{\text{LL}}}$
Shortest path $(h_1,\ ...,\ h_p,\ h) \leftarrow$ `Dijkstra`(h)
Return: h_1

Table 2. **PROMO Hyperparameters (Reacher)**

	Final Value	**Range Tried**
n_{steps} (expl. steps per episode)	8000	2000 - 20000
n_{bins} (landmark gen grid)	50	50
ϵ (finishing condition)	0.001	0.001, 0.005
SVM RBF kernel ν	0.01	0.01 - 0.1
SVM RBF kernel γ	0.1	0.01 - 0.1
LSTM hidden dim	64	64
LSTM batch size	100	100, 100, 500
LSTM num train epochs	500	200, 500
LSTM train steps	500K	500K, 1M
π_{LL} Gaussian action noise σ	2.	0.1 - 2
π_{LL} learning rate	0.001	0.001
π_{LL} batch size	256	256
π_{LL} train steps	500K	500K

Table 3. **Hyperparameters for DHRL and BEAG.** We experimented with various hyperparameter configurations, as well as different numbers of landmarks.

	DHRL	**BEAG**
initial episodes without graph planning	75	-
gradual penalty	1.5-5.0	-
high-level train freq	10	-
failure count threshold τ_n	-	100,200,300,400
failure condition threshold τ_t	-	1,3,5,7
number of landmarks	300-600	36-256
hidden layer	(256, 256)	(256, 256)
actor lr	0.0001	0.0001
critic lr	0.001	0.001
target network soft update rate τ	0.005	0.005
discount factor γ	0.99	0.99
batch size	1024	1024
target update freq	10	10
actor update freq	2	2